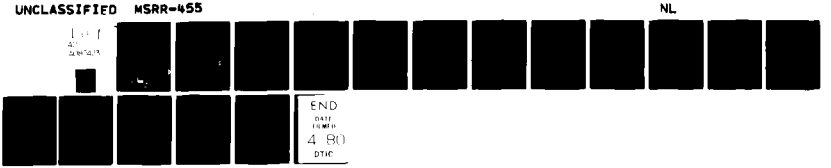
AD-A082 423 CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENC--ETC F/G 12/2
THE NON CANDIDATE CONSTRAINT METHOD FOR REDUCING THE SIZE OF A --ETC(U)
FEB 80 A P SETHI, G L THOMPSON N00014-75-C-0621
UNCLASSIFIED MSRR-455 NL



END
DATE FILMED
4 80
DTIC

LEVEL II

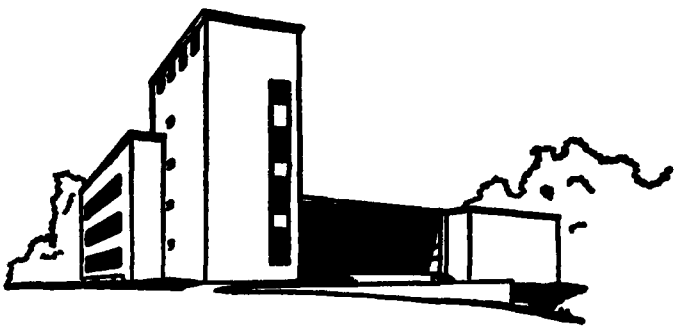
AD A082423

Carnegie-Mellon University

PITTSBURGH, PENNSYLVANIA 15213

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



DTIC
ELECTE
MAR 3 1 1980
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DDC FILE COPY

80 3 28 020

W.P. 452-79-80

(9) Management Sciences Research Report No. 455

(14) MSRR-455, WP-52-79-8

(6) THE NON CANDIDATE CONSTRAINT METHOD FOR REDUCING THE SIZE OF A LINEAR PROGRAM

by

(10) Awanti P. Sethi

and

Gerald L. Thompson

(12) 17

(11) February 1980

DTIC ELECTE MAR 3 1 1980 S D A

(15)

This report was prepared as part of the activities of the Management Sciences Research Group, Carnegie-Mellon University, under Contract N00014-75-C-0621 NR 047-048 with the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Management Sciences Research Group
Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

403426

A

THE NON CANDIDATE CONSTRAINT METHOD FOR REDUCING THE SIZE OF A LINEAR PROGRAM

by

Awanti P. Sethi and Gerald L. Thompson

ABSTRACT

A non candidate constraint in a linear program is one which never contains a pivot element during the course of solving the problem. Discovering non candidate constraints is computationally costly since their discovery, in general, depends on the actual sequence of pivots used. Knowing which constraints are non candidate is of great computational benefit since they need not be kept in updated form. Our experience indicates that from 50 to 80 percent of the constraints in randomly problems are non candidates at least part of the time.

In this paper we present a "learning" approach to the identification of non candidate constraints. At each iteration we determine which constraints can potentially be pivotal; these are candidate constraints and all others are non candidate constraints on that step. On proceeding with the simplex method we update only the candidate constraints. If a non candidate constraint becomes candidate on a later step, we update it and add it to the candidate list.

Although the constant checking of constraints to see whether they are changing from being candidate to non candidate is computationally costly, we obtain the computational benefit of having to keep in updated form a much smaller tableau.

The net benefit of using this strategy is positive and results in a 25 to 50 percent reduction in total computation time. This paper describes the method in detail and gives computational results of testing it on a series of randomly generated problems.

THE NON CANDIDATE CONSTRAINT METHOD FOR REDUCING THE SIZE OF A LINEAR PROGRAM

by

Awanti P. Sethi and Gerald L. Thompson

1. Introduction.

Thompson, Tonge, and Zionts [4,6] defined a redundant constraint in a linear programming problem as one which could be dropped without changing the primal constraint set. Most linear programs have some redundant constraints, and it would be desirable to be able to identify them. However, the discovery of redundant constraints is computationally extremely costly.

A much weaker concept is that of a non-candidate constraint, to be defined precisely in Section 2. Intuitively a permanently non-candidate constraint is one which never becomes pivotal, that is, never contains a pivot element, during the course of solving a given linear program. Such constraints may, but need not, be redundant in the strong sense of [4]. But they are never used during the course of solving a problem, so that keeping them in updated form is of no value.

The discovery of permanently non candidate constraints is computationally even more difficult than the discovery of redundant constraints, since, in general, they depend on the choice of a specific sequence of pivots. However, we have found that from 50 to 80 percent of the constraints in randomly generated problems are sometimes non candidate constraints, see Section 4, so that their discovery can be of great computational benefit.

In this paper we present a "learning" approach to the identification of non candidate constraints. At each iteration we determine which constraints can potentially be pivotal; these are the candidate constraints; all others are non candidate constraints on that step. In proceeding with the simplex iterations

we update only candidate constraints, on the assumption that non candidate constraints on that step won't become candidates later. In case an error is made, and a previously non candidate becomes a candidate constraint on a later iteration, we update it, and add it to the candidate list.

Although the constant checking of constraints to see if they are changing from being candidate to non candidate constraints or vice versa is costly, there is the computational benefit of having a much smaller tableau to update at each pivot. The net benefit of using this strategy is positive and results in 25-50 percent reduction in total computation time, see Section 4.

2. Basis for the Method.

The non candidate constraint method is merely a modification of the standard simplex method, no new theoretical results are needed to establish the correctness of the approach. In order to explain it we set up some standard linear programming terminology.

Consider a linear programming problem stated form

$$\begin{aligned} &\text{Maximize } \{z = cx\} \\ &\text{Subject to } Ax = b \\ &\qquad x \geq 0 \end{aligned} \tag{1}$$

where A is $m \times n$, b is $m \times 1$, c is $1 \times n$, and x is $n \times 1$. We assume slack variables have been included in the definition of A, and that the problem has been transformed (by well-known techniques) so that $b \geq 0$. The simplex tableau at the kth iteration is denoted by

$A^{(k)}$	$b^{(k)}$
$c^{(k)}$	$z^{(k)}$

(2)

where initially

$$A^{(0)} = A, \quad b^{(0)} = b, \quad c^{(0)} = -c, \quad Z^{(0)} = 0. \tag{3}$$

We define the following sets of indices

$$I = \{1,\ldots,m\} \tag{4}$$

$$J = \{1,\ldots,n\} \tag{5}$$

$$J^{(k)} = \{j \mid j \in J \text{ and } c_j^{(k)} < 0\} \tag{6}$$

$$R_j^{(k)} = \{i \mid i \in I \text{ and } \frac{b_i^{(k)}}{a_{ij}^{(k)}} = \underset{a_{hj}^{(k)} > 0}{\text{Minimum}} \frac{b_h^{(k)}}{a_{hj}^{(k)}} \} \tag{7}$$

Clearly, $J^{(k)}$ is the set of indices of potential incoming variables at step k, while $R_j^{(k)}$ is the set of potential outgoing rows if column j is chosen to come in.

We can now define the set of candidate constraints at iteration k as

$$S^{(k)} = \{i \mid i \in R_j^{(k)} \text{ for some } j \in J^{(k)}\} . \tag{8}$$

The set of non candidate constraints at iteration (k) is then obviously

$$I - S^{(k)} . \tag{9}$$

In this paper we will use the maximum objective change rule [3, 5] , which we will call the maximum z-change rule. To find it we calculate

$$\delta^{(k)} = \underset{j \in J^{(k)}, i \in R_j^{(k)}}{\text{Maximum}} \left(\frac{b_i^{(k)}}{a_{ij}^{(k)}} \right) (-c_j^{(k)}) \tag{10}$$

and determine the indices of the incoming variable j and outgoing variable i so that (1) holds and then

$$z^{(k+1)} = z^{(k)} + \delta^{(k)}. \tag{11}$$

Although a large amount of time is needed to determine the pivot giving the maximum z-change, a net improvement in computational performance has been found when it is used to solve moderate sized problems [5]. In this paper we will show that still further improvements can be obtained by using the maximum z-change rule in conjunction with the non candidate method.

By a partial pivot for a given choice of pivot row i and pivot column j, we shall mean the calculation of the updated right hand sides $b^{(k+1)}$ from $b^{(k)}$. We will say that constraint i is violated if $b_i^{(k+1)} < 0$. Because of the choice of pivot row i by the minimum ratio rule (7) and (10), it is clear that no non candidate constraint at step k is ever violated at step $k+1$ that is,

$$b_i^{(k+1)} \geq 0 \quad \text{for} \quad i \in I - S^{(k)}. \tag{12}$$

However a constraint which is a non candidate at step k could be violated at step $k+2, k+3, \ldots$.

In the non candidate method described in the next section, the simplex tableau is updated at iteration k only for the constraints which are candidate at iteration k. Since from 50 to 80 percent of all constraints in randomly generated problems are non candidate, this results in considerable computational savings, see Section 4 .

3. The Non Candidate Algorithm.

The flow chart for the non candidate simplex algorithm is shown in Figure 1. The initialization steps of finding the set $S^{(0)}$ of candidate constraints and the first maximum z-change pivot are given in boxes 1 and 2. They can be completed by making one pass through the original data of the problem. Step 3 is the pivot

step which updates the rows of the simplex tableau restricted to the candidate constraints. The rows corresponding to the non candidate constraints are not updated. In box 4 a test is made to see if the current tableau is optimal; if it is the algorithm stops; otherwise it continues.

In box 5 of Figure 1, the computer is instructed to make one pass through the updated tableau at the kth step which is restricted to the rows in $S^{(k)}$; during the data pass all non candidate constraints are removed from $S^{(k)}$ to form the set $S^{(k+1)}$. During this data pass it is possible to also carry out the instruction in box 6 of finding the maximum z-change pivot in the tableau restricted to $S^{(k+1)}$.

In box 7, the computer is instructed to perform a partial pivot on $b^{(k)}$ restricted to the set $S^{(k+1)}$; at the end of the partial pivot $b^{(k+1)}$ restricted to the same set and hence $x^{(k+1)}$ will be available. As instructed in box 8, the proposed solution, $x^{(k+1)}$, is then substituted into the non candidate constraints in the set $I - S^{(k)}$ to see if any of them is violated. If any such constraint is found to be violated, the computer is instructed in box 9 to update that constraint, and then go to box 5 to try again to find a maximum z-pivot and a new $x^{(k+1)}$. This process is repeated until eventually a proposed $x^{(k+1)}$ will be found which does not violate any non candidate constraint. When this happens, the computer goes from box 8 to box 10 where k is updated; and from there it goes back to box 3 to continue with the pivoting process.

Since the algorithm outlined is simply a reorganization of the standard simplex method, all of the anti-cycling techniques, and finiteness proofs for the latter hold for the non candidate method as well.

4. Computational Experience.

Table 1 gives the time comparison between the non candidate greatest z-change method and the greatest z-change method in solving a series of 19

randomly generated problems with two different densities of problem data. Since the same entry criterion was used in each method the sequence of pivots was identical in each. Hence the differences in solution times can be attributed entirely to the relative efficiencies of the two methods. We found that the non candidate method is more effective on problems with higher density, and also on larger problems.

Table 2 gives a computational comparison of the non candidate method and the ordinary simplex method on two sets of randomly gen' ated problems. Note that the number of iterations is smaller for the non candidate method due to its greatest z-change entry criterion. The ratio of times for the two methods are close to the ratio of number of iterations, which indicates that the savings in time from not updating non candidate constraints approximates the additional time needed for the greatest z-change calculations.

We have so far not tried to compare the two methods on actual problems derived from real applications. However, the results so far on these randomly generated problems are very encouraging.

5. Variations and Extensions of the Method.

Many variants of the method described in this paper have occurred to us which we have not yet had time to test. We discuss some of them here.

(a) Connections with the GUB Method. The generalized upper bound (GUB) [6] method handles constraints which are sums of variables outside of the tableau. Usually such GUB constraints are also non candidate in our sense. When they are candidate constraints and are actually chosen to be pivotal, they are not re-introduced into the tableau as we do; instead special pivoting rules are added to the simplex algorithm. Perhaps such special pivoting rules could be extended to more general kinds of constraints.

(b) Intermittent candidate checking. Instead of checking for candidate constraints at each iteration as we do in box 5 of Figure 1, we could instead, perform this check only once every sth iteration, s = 5, say, where the danger is that a constraint could then become violated, and dual feasibility lost. This could be corrected by performing some dual pivots. In this way the method could become a kind of primal-dual algorithm. Only actual computational tests could show whether, and for what values of s, this method would be good.

(c) Extension to large problems. Most large linear programming problems are very sparse initially, but the tableau can fill in during the course of the computation. Hence the non candidate method is a potentially valuable way of keeping down the size of the working tableau, and/or the size of the basis inverse matrix. The difficulty is in preforming the check for candidate constraints. To this end we propose that the original data be stored on two tapes, once in row order, and once in column order. Then the tape with row ordered data is used to check for candidates, and the tape with column ordered data used to check for incoming variables. We believe that the computations could be organized so that both of these steps could be done efficiently. If the number of candidate constraints is as small as we found in Table 2, then dramatic improvements in the solution of large linear programs might be possible using this method.

6. Conclusions.

In this paper we have defined the concept of candidate and non candidate constraints, and have demonstrated computationally that a modification of the simplex method can be made to take advantage of the fact that non candidate constraints need not be updated. The method assumes that a non candidate constraints at step k will remain so. If an error is made and such a constraint becomes a candidate at a later step then the constraint is updated and put back

into the tableau. Significant computational savings are possible by using this method. Extensions of the method to find the solution to large problems were sketched.

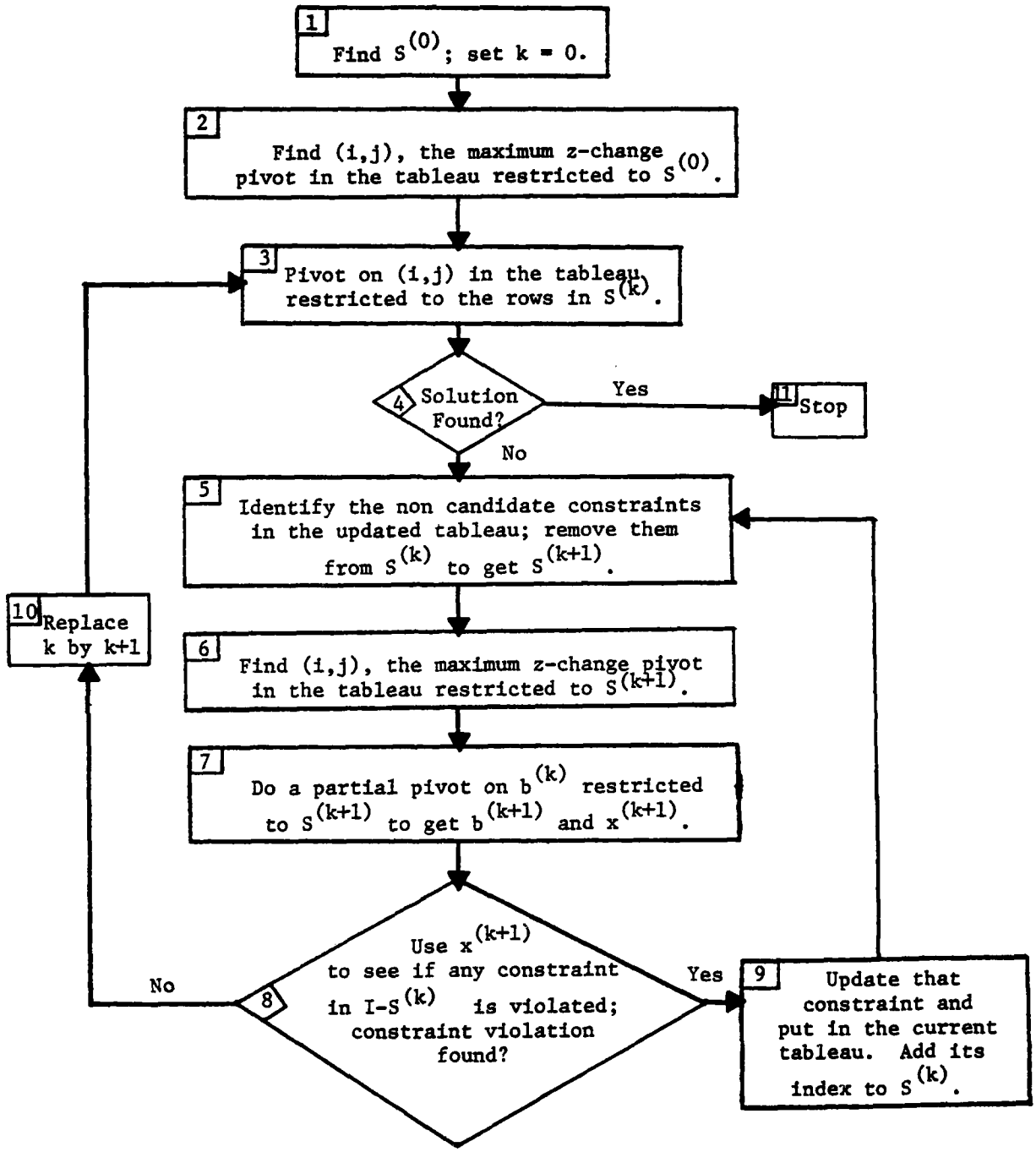


Figure 1. Flow chart for non candidate simplex algorithm.

Rows	Columns	Greatest z-change non candidate Time (Secs)	Greatest z-change simplex Time (Secs)	Ratio of Times
47	49	3.27	5.19	.63
62	130	14.71	19.57	.75
63	130	16.69	24.69	.67
83	94	17.48	22.67	.77
99	100	20.81	38.34	.54
100	99	13.59	19.68	.69
142	55	10.79	18.67	.58
150	180	66.68	134.25	.50
19	16	.20	.24	.87
20	20	.30	.35	.86
45	120	10.01	11.94	.84
59	92	10.87	12.62	.86
67	132	14.60	19.90	.73
71	40	3.75	5.20	.72
79	100	12.38	19.02	.65
92	74	12.24	18.99	.64
100	87	14.24	24.41	.58
125	70	19.18	33.93	.56
172	26	10.28	16.07	.64

Table 1

Comparison of greatest z-change non candidate and simplex methods. The sequence of pivots was the same for both methods, so that the ratio of times indicates the relative efficiency of the two methods.

			Non Candidate Method				Simplex Method		Ratios	
				Candidate Constraints						
Rows	Cols	Probs	Ave.No. Iter.	Ave No	%	Ave. Time	Ave.No. Iter.	Ave Time	Iter.	Times
20	100	3	25	13.33	.67	1.64	34	1.71	.74	.96
40	100	3	43	20.66	.52	6.31	78	8.82	.55	.72
60	100	2	49.5	20.50	.34	9.56	90	17.42	.55	.55
80	100	1	59	28	.35	16.98	200	56.99	.30	.30
100	100	1	57	26	.26	21.11	95	36.55	.60	.58
150	100	1	68	29	.19	42.94	139	105.31	.49	.41

Problem Set 1. Coefficients were generated at random with 50% positive, 10% negative, and 40% zero.

			Non Candidate Method				Simplex Method		Ratios	
				Candidate Constraints						
Rows	Cols	Probs	Ave.No. Iter.	Ave No	%	Ave Time	Ave.No. Iter.	Ave Time	Iter.	Times
50	100	3	27	14.7	.49	2.63	42.7	3.55	.63	.74
50	100	2	31	20	.40	5.25	57.5	9.20	.54	.57
90	130	1	58	25	.28	20.22	71	31.75	.80	.67
130	80	1	48	23	.18	24.89	84	48.74	.57	.51

Problem Set 2. Coefficients were generated at random with 65% positive, 15% negative, and 20% zero.

Table 2

Comparison of the greatest z-change non candidate method and the simplex method with most negative reduced cost entry criterion.

References

[1]. Charnes, A., "Structured Sensitivity Analysis in Linear Programming and an Exact Product Form of Inverse," Naval Research Logistic Quarterly, 15 (1958), 517-522.

[2]. Jeroslow, "The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement," Discrete Mathematics, 4 (1973), 367-377.

[3]. Lemke, C., "The Dual Method of Solving the Linear Programming Problems," Naval Research Logistics Quarterly, 1 (1954), 36-47-

[4]. Thompson, G. L., S. Tonge and S. Zionts, "Techniques for Removing Non-Binding Constraints and Extraneous Variables from Linear Programming Problems," Management Science, 12, (1966), 588-608.

[5]. Wolfe, P. and L. Cutler, "Experiments in Linear Programs (Run 41)" in the book Recent Advances in Mathematical Programming, edited by R. L. Graves and P. Wolfe, New York, McGraw Hill, 1963.

[6]. Zionts, S., Linear and Integer Programming, Prentice Hall, New Jersey, 1974.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: MSRR NO. 455
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): THE NON CANDIDATE CONSTRAINT METHOD FOR REDUCING THE SIZE OF A LINEAR PROGRAM

5. TYPE OF REPORT & PERIOD COVERED: Technical Report February 1980

6. PERFORMING ORG. REPORT NUMBER: MSRR 455

7. AUTHOR(s): Awanti P. Sethi
G. L. Thompson

8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0621

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Graduate School of Industrial Administration
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR 047-048

11. CONTROLLING OFFICE NAME AND ADDRESS: Personnel and Training Research Programs
Office of Naval Research (Code 458)
Arlington, Virginia 22217

12. REPORT DATE: February 1980

13. NUMBER OF PAGES: 11

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

simplex method

non candidate constraint simplex method

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A non candidate constraint in a linear program is one which never contains a pivot element during the course of solving the problem. Discovering non candidate constraints is computationally costly since their discovery, in general, depends on the actual sequence of pivots used. Knowing which constraints are non candidate is of great computational benefit since they need not be kept in updated form. Our experience indicates that from 50 to 80 percent of the constraints in randomly problems are non candidates at least part of the time.

(OVER)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

In this paper we present a "learning" approach to the identification of non candidate constraints. At each iteration we determine which constraints can potentially be pivotal; these are candidate constraints and all others are non candidate constraints on that step. On proceeding with the simplex method we update only the candidate constraints. If a non candidate constraint becomes candidate on a later step, we update it and add it to the candidate list.

Although the constant checking of constraints to see whether they are changing from being candidate to non candidate is computationally costly, we obtain the computational benefit of having to keep in updated form a much smaller tableau.

The net benefit of using this strategy is positive and results in a 25 to 50 percent rduction in total computation time. This paper describes the method in detail and gives computational results of testing it on a series of randomly generated problems.